UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 09/620,714 | 07/20/2000 | Luke Matthew Browning | AUS9-2000-0277-US1 | 3354 |

35525 7590 03/09/2004

DUKE W. YEE
CARSTENS, YEE & CAHOON, L.L.P.
P.O. BOX 802334
DALLAS, TX 75380

| EXAMINER | |
|---|---|
| STEELMAN, MARY J | |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2122 | 13 |

DATE MAILED: 03/09/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

| Office Action Summary | Application No. 09/620,714 | Applicant(s) BROWNING ET AL. | |
|---|---|---|---|
| | Examiner Mary J. Steelman | Art Unit 2122 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on *06/02/03,09/03/03,12/24/03*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) *1-29* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-29* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *10 October 2003* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. §§ 119 and 120**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

13)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

a) ☐ The translation of the foreign language provisional application has been received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

**Attachment(s)**

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____ .
4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☒ Other: *Copy of accepted drawings.*

## DETAILED ACTION

1. Claims 1-29 are pending.

2. In view of Applicant's Brief, filed 12/24/2003, Examiner withdraws prior Final Rejection, mailed 07/30/2003.

3. Per Applicant's request in Amendment A, dated 30 May 2003, claims 1, 6, 13, and 18 have been amended and claims 26-29 have been added. Applicant's arguments in "Response to Final Office Action" dated 3 September 2003 and "Appeal Brief", dated 24 December 2003 have been considered.

### *Drawings*

4. Formal drawings submitted 10 October 2003 have been accepted by Examiner.

### *Claim Rejections - 35 USC § 102*

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

6. **Claims 1-6, 11-18, & 23-29** are rejected under 35 U.S.C. 102(e) as being anticipated by U.S. Patent 6,240,529 to Kato.

**Per claims 1, 13, & 25:**

Kato disclosed a method, apparatus (col. 6, line 11) and a program recorded on a recording medium (col. 7, line 5) for debugging a process and storing state information.

-initiating debugging of the process; (Col. 7, line 15, "When debugging is started..."); -saving a process state in response to a first event to form a stored process state; (Col. 7, lines 65-66, "storing a debugged state into a file if a certain event occurs." (a first event)); -retrieving the stored process state in response to a predefined event; and reinitiating debugging from the stored process state. (Col. 8, lines 29-35, "When a state restoration command is issued, storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list. If a file to be restored is selected from the information, then state restoration unit reads in the file (retrieves) and restores (restore at #333, FIG 5B / reinitiates at 'A' exiting from #333 and proceeding to FIG 5A) debugged state." (emphasis added) See FIG. 3, Execution units #108 & #109 (processor) and storage file, #118 (memory). FIG. 4, Execution upon restoration using saved / checkpointed state. FIG. 5B, #333.)

**Per claims 2 and 14:**

-first event occurs periodically. (Col. 6, lines 14-19, "storage...for...debugged state...can be designated at an arbitrary point of time...")

**Per claims 3 and 15:**

-process state is saved in a checkpoint data structure. (Col. 10, lines 30-32, "debugged state is stored, a state storage file...")

**Per claims 4 and 16:**

-checkpoint data structure is a checkpoint file. (Col. 10, lines 30-32, "debugged state is stored, a state storage file... (checkpoint data structure / checkpoint file)")

**Per claims 5 and 17:**

-checkpoint data structure includes a process descriptor for the process. (FIGS. 4 & 6, State Storage File Name, and col. 5, lines 43-45, "storing a state storage file name and the situation upon the storage in a correlated condition into a storage situation management file...")

**Per claims 6 and 18:**

-initiating debugging of the process; (Col. 7, line 15, "When debugging is started...")

-saving a process state in response to a first event to form a stored process state; (Col. 7, lines 65-66, "storing a debugged state into a file if a certain event occurs." Also see FIG. 5A, #312. If it is requested to store state, then at #313 state is stored into a file at FIG 4.);

-retrieving the stored process state in response to a second event; (See FIG. 5A. At #304 option 'B' can lead to FIG. 5B and the retrieval of a stored process state at #333.Col. 8, lines 29-35, "When a state restoration command is issued, storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list. If a file to be restored is selected from the information, then state restoration unit reads in the file (retrieves) and restores (reinitiates) the debugged state." FIG. 3,

Execution units #108 & #109 (processor) and storage file, #118 (memory). FIG. 4, Execution upon restoration. FIG. 5B, #333.)

-reinitiating debugging from the stored process state, wherein the process has control over at least one child process and the process state includes a process descriptor for each of the at least one child process. (FIG. 5A. Note 'A' cycles from the 'end' of the FIG. 5A back to the 'beginning' (reinitiating). Kato disclosed (col. 8, lines 16-19), "a function to manage a situation upon storage of a debugged state is further added. (child process) In this connection, storage situation management unit is additionally provided for state storage unit. (checkpoint)" Also note, col. 6, lines 30-34, "...the means for managing a situation when the debugged state is stored (checkpoint), a state storage file with which an intended debugged state can be restored can be retrieved by referring to a list or trace information, and restoration processing of a debugged state is facilitated (another child process)." The debug process calls many child processes. In calling child processes, the debug process certainly knows the "process descriptor" for each of the at least one child processes.

**Per claims 11 and 23:**

-the process state is saved when the program is in a stopped state. (Col. 7, lines 65-66, "storing a debugged state into a file if a certain event occurs.")

**Per claims 12 and 24:**

-the stopped state is at a breakpoint. (Col. 7, lines 37-38, "A break point at which execution of a program is interrupted..." Also, col. 10, lines 20-26.)

**Per claim 26:**

-the first event is a breakpoint and the predefined event is a checkpoint, and further comprising the step of repeatedly running between the checkpoint (event registered) and the breakpoint for a plurality of times. (See FIG. 5A, #304, 'B', and #314, and Col. 9, lines 15-36, "Then, it is checked whether or not a state corresponding to one of events registered (checkpoint) by the user is satisfied...If a request for a break (breakpoint) is generated in response to the detection of the event (314), then the execution flag is changed to OFF (316)." (first event is a breakpoint) After cycling back via 'A' to #304, if the execution flag is set to OFF, a break point may be indicated, at which point the 'B' option is followed (to FIG. 5B). In FIG. 5B at #330 a checkpoint file is saved or at #322 AND #333, a checkpointed file is selected (See FIG. 4) and state is restored to the debug process using the information in the checkpoint file. Execution continues to 'A' of FIG. 5A. )

**Per claim 27:**

-variable values are automatically modified after retrieving the stored process state for the checkpoint. (Col. 8, lines 33-35, "If a file to be restored is selected from the information, then state restoration unit reads in the file (modify values) and restores the debugged state." Also see FIG. 3, #117, state restoration unit.)

**Per claim 28:**

-initiating a debug process; (Col. 7, lines 15.)

-creating a child process from the debug process; (Col. 10, lines 43-46, "...the storage situation management unit...additionally has a function of recording a debugged state storage timing into trace information." Note FIG. 5B. Seven processes are forked off of FIG, 5A, #304)

-saving a process state of the child process in response to a first event, to form a stored process state; (Col. 10, lines 49-51, "...information of the state storage file name and so forth is stored also into a buffer (saving state of child process) for trace information storage." Also see FIG. 6)

-retrieving the stored process state in response to a second event; executing the child process using the stored process state (See FIG. 5B, #333 & 'A' for re-executing). (Col. 11, lines 19-29, "...the debugged state storage file name is recorded into 407 (FIG. 6) at a timing at which the debugged state is stored...the retrieval means which refers to trace information is additionally provided, and specification of a debugged information file to be restored is facilitated by confirming a state storage file, a type of an execution instruction upon production of the file, a state of memory and so forth.")

**Per claim 29:**

-tracing a process by a debugger; (Col. 7, lines 35-37, "Debugger unit in debugging apparatus performs processing regarding a debugging function for breaking, tracing or the like.")

-saving a process state of the traced process and a process state of another process that is not being traced by the debugger; (Col. 7, lines 35-47 and 65-66, "...an instruction execution situation, a memory access situation and so forth at the point of time are recorded..." Tracing is disclosed by Kato in Embodiment 2 of the invention. See col. 10, lines 33-36. Multiple process states are saved, some may be from Embodiment 1 (not traced) and some from Embodiment 2 (traced).)

-retrieving the saved process states; (Col. 8, lines 29-35, "When a restoration command is issued...")

-reinitiating debugging of the process using the retrieved process states. (Col. 8, lines 29-35, "When a state restoration command is issued, storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list. If a file to be restored is selected from the information, then state restoration unit reads in the file and restores the debugged state." Also col. 10, lines 33-36, "...an example wherein more precise retrieval of a debugged state to be restored is allowed by combining them with another debugging function." Also, FIGS. 5A and 5B, reinitiate debugging at 'A'.)

***Claim Rejections - 35 USC § 103***

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. **Claims 7-9 and 19-21** are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent 6,240,529 to Kato, and further in view of U.S. Patent 5,560,009 to Lenkov et al. Kato disclosed an invention to debug and save state to a file, allowing resumption. Kato failed to provide information on data type descriptors, instance descriptors, and data block corresponding to an instance descriptor.

However, Lenkov described an invention that generated symbolic debug information, including creating a debug data structure (col. 6 , line 62), and (col. 7, lines 33-48) including

various information...debug name and type tables, a table of name strings, and object file symbol table. Col. 8, lines 39-46, "The core object file class is used to read common object file data structures...includes access routines to look up symbols in symbol tables... (Col. 16, lines 57-67), "the dtab class includes member for holding state during iterations...The classes, data structures, variables and functions which comprise the dtab... (Col. 17, lines 55-60) blocktab is a derived class of dtab. Blocktab is the base class of all source blocks... (Col. 25, lines 46-51) The output object file interface creates a new object/debug file and stores...debug information." Also, (Col. 26, lines 20-21) "output object file interface is implemented in the form of classes...instantiations..." Also, (Col. 26, lines 56-58) "the preprocessor creates a source file descriptor table, a procedure descriptor table, a class descriptor table..."

Therefore, it would have been obvious, to one of ordinary skill in the art, at the time of the invention, to modify Kato's invention to debug, store state information, and restore state, by including Lenkov's invention that processed debug information, because it arranges symbolic debug information for storage and retrieval in a meaningful manner for efficient diagnostics..

9. **Claims 10 and 22** are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent 6,240,529 to Kato, and further in view of U.S. Patent 6,412,106 to Leask et al..
Kato disclosed an invention to debug and save state to a file, allowing resumption. Kato failed to provide information regarding modifications prior to resuming a debug process.

However Leask disclosed a debugging invention that allows for modifying values while the program is suspended.

Therefore, it would have been obvious, to one of ordinary skill in the art, at the time of the invention, to modify Kato's invention to debug and store state, by including the features in Leask's invention that allow for modifying values because it would create a quicker interactive debug session, saving time by not requiring a start from the beginning of a program.

**Per claims 10 and 22:**

-modifying at least one register or memory variable before resuming debugging from the stored process state. (Leask, col. 12, lines 55-56, "capability to modify values stored in variables, while the application program is suspended.")

10. ***Response to Arguments***

**Applicant's arguments regarding claims 1-5, 7-17, and 19-25 (GROUP I) are as follows:**

(A) I. On page 6, 4th paragraph, of Applicant's Brief, received 31 December 2003; Applicant cites that "Kato's restoration is responsive to a state restoration command *currently* being issued. This does not teach or otherwise disclose that claimed step of retrieving the stored process state *in response to a predefined event.* The cited reference requires a *manual restoration* of a state storage file as selected by a user from a list of existing files...this retrieval is *responsive to a user input command.*" (Emphasis, as added by Applicant.)

II. Additionally, on page 8, 2nd paragraph, Applicant cites, "...the teachings of the Kato reference are directed to assisting the user in manually selecting from one of many state storage files to be used when the user manually issues a state restore command" which "cannot

reasonably be construed to read on the claimed feature of retrieving the stored process state in response to a predefined event."

**Examiners Response to the Applicant's arguments is as follows:**

I. First, Kato indeed teaches the "predefined event", the "claimed step of retrieving the stored process state in response to a predefined event" at col. 8, lines 29-35, which states, "When a state restoration command (a predefined event) is issued.... state restoration unit reads in the file and restores the debugged state." (emphasis added)

II. Second, note that the plain language of the claim does not exclude and / or include automatic or manual user intervention in the restoration of state information. Thus there is no need to read 'automatic' as Applicant contends.

**Applicant's arguments regarding claims 6 and 18 (GROUP II) are as follows:**

(B) I. As noted on page 9, 2nd paragraph, "...the cited reference does not teach the claimed step of 'reinitiating debugging from the stored process state, wherein the process has control over at least one child process and the process state includes a process descriptor for each of the at least one child process." Kato does not teach "wherein the process has control over at least one child process and the process state includes a process descriptor for each of the at least one child process." Applicant claims (page 10, 1st paragraph) that invention "provides process state information for both (1) the process and (2) the at least one child process."

II. Additionally, on page 10, 1st paragraph, Applicant states, "The cited reference only alludes to maintaining debug capabilities for a single running process." Whereas, Applicant's invention provides "debug capabilities in an environment with multiple concurrently running processes."

**Examiners Response to the Applicant's arguments is as follows:**

I. Applicant's argument that invention "provides process state information for both (1) the process and (2) the at least one child process" is not in the claim language of claim 6, nor claim 18. Furthermore, the claim language fails to state the parent relationship of the "child process." It is not clear from the claim language, that the debug process is the parent of a child debug process.

Kato disclosed the limitations of claim 6 (col. 8, lines 16-19), "a function to manage a situation upon storage of a debugged state is further added. (a child process) In this connection, storage situation management unit is additionally provided for state storage unit." See FIG. 5A, (col. 6, lines 51-52) "a flow chart illustrating a processing flow (a process) of a debugging method..." (emphasis added) Also note, col. 6, lines 30-34, "...the means for managing a situation when the debugged state is stored, a state storage file with which an intended debugged state can be restored can be retrieved by referring to a list or trace information (See FIG. 4), and restoration processing of a debugged state is facilitated (another child process)." (emphasis added) The debug process (FIG. 5A) calls many child processes. At FIG. 5A, #304->B, processes called include (See FIG. 5B): #322 – 'BACK EXECUTE ONE INSTRUCTION', one

by one until a desired point is reached, #322 & #333 – Select from a state storage file list for a checkpointed state and restore a debugged state. In calling child processes, the debug process certainly knows the "process descriptor" for each of the at least one child processes.

II. "debugging a process in a multi-process environment" is not in the claim language of claims 6 and 18. The argument is moot.

**Applicant's arguments regarding claims 26 and 27 (GROUP III) are as follows:**

(C) As noted on page 10, 3rd paragraph, claim 26 and dependent claim 27 "recite that the predefined event (to which the retrieval of the stored process state of Claim 1 is responsive to) is a checkpoint. Applicant further notes, "by providing the ability to restore an image of the process being debugged in response to an event- and in this case a restore checkpoint – the debugger may advantageously be instructed to automatically run between a checkpoint and a breakpoint repeatedly....using a plurality of register or memory variable values to assist in debugging the process."

Additionally, on page 10, lines 23-30, Applicant notes, "The cited reference does not teach the claimed steps of "retrieving the stored process state in response to a predefined event"... "wherein the predefined event is a checkpoint" To the contrary, Kato shows in FIG. 5A, #309 and #314, that Kato's invention "checks whether an event has occurred and if so, processing of the action registered in connection with the event is performed", but fails to

mention the "specific action of retrieval of a stored process state in response to this event checking."

**Examiners Response to the Applicant's arguments is as follows:**

I. Applicant has stated on page 10, lines 19-20, "...the debugger may advantageously be instructed to automatically run between a checkpoint and a breakpoint repeatedly." The word "automatically" (page 10, line 20 of Appeal Brief) is not in the claim language. To the contrary, claim 26 recites, "...repeatedly running between the checkpoint and the breakpoint for a plurality of times." Kato discloses this feature in FIG. 5A and FIG. 5B. At #314 of FIG. 5A, it is determined whether there exists a request for a break (first event is a breakpoint). After cycling back via 'A' to #304, if the flag is set to 'off', a break point is indicated, at which point the 'B' option is followed (to FIG. 5B). In FIG. 5B, at #329 an event (predefined event is a checkpoint) is checked as to whether to store a checkpoint file or retrieve a checkpoint file, at #330 a checkpoint file is stored, at #322 and #333, a checkpointed file is selected (See FIG. 4) and state is restored to the debug process using the information in the checkpoint file. Execution continues to 'A' of FIG. 5A.

II. The claim language states, "variable values", not "using a plurality of register or memory variable values to assist in debugging the process." Kato disclosed (col. 8, lines 33-35), "If a file to be restored is selected from the information, the state restoration unit reads in the file and restores the debugged state." (Variable values are automatically modified as the restoration unit reads in the file to reset the state of the process.)

III. Kato disclosed the "specific action of retrieval of a stored process state in response to this event checking" at (See FIG. 5B) col. 10, lines 3-9, "If a state restoration command is inputted (331) (in response to event), a file to be restored is selected (332) from within the storage file list (FIG. 4) based on the storage situation management file which manages situations upon storage of a debugged state...and the selected file is read in to restore (retrieval of a stored process state / variable values are modified) the debugged state." (emphasis added)

IV. Applicant argues that Kato's invention "checks whether an event has occurred and if so, processing of the action registered in connection with the event is performed", but fails to mention the "specific action of retrieval of a stored process state in response to this event checking." However, this is not in the claim language. See claims 26 and 27. Furthermore, note at #333 of FIG. 5B that the stored process state is used to restore the debug process to a checkpointed state.

**Applicant's argument regarding claim 28 (GROUP IV) is as follows:**

(D) As cited on page 12, last paragraph, of Appeal Brief, Applicant claims, "reference does not teach the claimed steps of "creating a child process from the debug process", "saving a process state of the child process" (the child process having been created from a debug process) and "executing the child process using the stored process state"."

Additionally, page 13, line 3, recites, "...two processes – a debug process and a child process (which is created from the debug process)."

**Examiners Response to the Applicant's argument is as follows:**

Claim language fails to clarify a (1) parent debug process and (2) a child debug process created from the parent debug process. A relationship is not noted. Examiner reads the claim language to state only that a child process (of some type) is created. Claim language fails to indicate, "that the child process (created from the debug process) is the object of debugging."

**Applicant's argument regarding claim 29 (GROUP V) is as follows:**

(E) I. As noted on page 14, 4th paragraph, of Appeal Brief, "The cited reference does not teach or otherwise disclose saving a process state of a traced process and a process state of another process that is not being traced."

Following on page 15, 1st and 2nd paragraphs, Applicant notes Kato, "col. 7, lines 35-47 generally discusses a debugging apparatus that has the ability to set a breakpoint and perform tracing when a certain event occurs..." Applicant claims that these "are not seen to be of importance with regards to claim 29 and specifically the saving of process states from two different processes."

II. Additionally, Applicant notes Kato, "col. 7, lines 65-66 merely describes an ability to store a debugged state into a file if a certain event occurs". "It does not teach or otherwise disclose saving a process state *of the traced process* and a process state of *another process that is not being traced by the debugger.* (Emphasis added by Applicant.) At best, it

teaches storing a single process state, whereas the claim is directed to storing two different process states from two different processes – with one state being traced and the other one not being traced."

**Examiners Response to the Applicant's arguments is as follows:**

I. Examiner's cited reference: -saving a process state of the traced process and a process state of another process that is not being traced by the debugger; (Col. 7, lines 35-47 and 65-66, "...an instruction execution situation, a memory access situation and so forth at the point of time are recorded...") Tracing is disclosed by Kato in Embodiment 2 of the invention. See col. 10, lines 33-36.

II. Additionally, see col. 10, lines 38-41, "...while a debugged state can be stored at the timing of detection of an event, also setting such that trace information is acquired upon detection of the same event is possible." (save state of traced process, save state of a process not traced) Also, col. 8, lines 29-32, "When a state restoration command is issued, storage situation management unit displays a list of currently existing state storage files based on storage situation management file. FIG. 4 shows an example of the list." Multiple process states are saved, some may be from Embodiment 1 (not traced) and some from Embodiment 2 (traced).

In conclusion:

Claim language does not clearly present a parent debug process and it's related child debug process.

Claim language does not state that the checkpoint / breakpoint cycle occurs automatically, without user intervention.

Claim language does not state "multiple concurrently running processes."

***Conclusion***

11. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

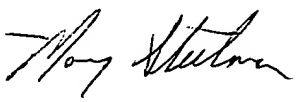
12. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

13. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Mary Steelman, whose telephone number is (703) 305-4564. The

examiner can normally be reached Monday through Thursday, from 7:00 A.M. to 5:30 P.M. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Dam can be reached on (703) 305-4552.

The fax phone number is (703) 872-9306 for regular communications and for After Final communications. Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-3900.

Mary Steelman



03/04/2004

TUAN DAM
SUPERVISORY PATENT EXAMINER